Upgrading C/C++ Applications from Win32 to Win64

March 2000

Joint White Paper by
Intel and MigraTEC, Inc.

Table of Contents

Introduction 2
Migrating Code: What's Involved 2
Why Migration Tools May be Necessary 3
Using the MigraTEC Migration Workbench 3
Inventory Process 3
Analysis Process 3
Remediation Process 3
In Summary 4
For More Information 4

Introduction

The benefits of upgrading your existing C/C++ applications from the Win32 environment to the Win64 environment are substantial. You can leverage applications that already have proven their value by providing a function in your organization while enhancing the usefulness of these applications with the added performance, scalability and reliability of the IA-64 platform. For this reason you're probably considering upgrading a number of such applications and are concerned about the effort involved.

To understand this effort, think of the upgrade process as a series of three steps:

1. Assessing complexity,

2. Developing a plan, and

3. Migrating your code.

In assessing complexity, you identify crucial third-party dependencies such as development tools and environments, source-code control mechanisms, and static/dynamic libraries. In developing a plan, you evaluate and select training methodologies and porting/development tools. In migrating your code, you use a methodical and iterative approach to examining and modifying the code base so it will run successfully in the Win64 environment.

In this white paper you will learn about tools and procedures you can use to simplify much of this work, especially in the area of migrating your code.

Migrating Code: What's Involved

While migrating your code, you're likely to encounter difficulties in three major areas: assignments and comparisons of pointer variables, declarations of return types, and APIs.

- **Pointer variables:** Pointer variables represent the biggest challenge in migrating code from Win32 to Win64. This is because of the difference between the data models used in the two environments. Under Win32, C/C++ code follows the ILP32 data model, which specifies that the three fundamental data types — *int*, *long*, and *pointer* — are implemented in 32 bits. Under Win64, C/C++ code follows the P64 data model, which specifies that int and long are implemented in 32 bits and pointer is implemented in 64 bits.

  In other words, under Win64 pointers are twice as long as ints or longs. To address this conflict, you need to identify all direct or implied assignments or comparisons between pointer variables and int or long variables. You also need to remove any casts that allow the compiler to accept assignment or comparison between pointer variables and int or long variables.

- **Return types:** You need to identify declarations of all variables, parameters and functions or return types that must change and redeclare them as one of the new size variant data types. These new data types correspond to existing data types in the Win32 environment. For example, you would declare a variable that was of type int in Win32 to be of type *int_ptr* (an integer the size of a pointer) in Win64.

- **APIs:** You also need to replace existing 32-bit APIs with new size variant APIs. For example, you would change the 32-bit API *SetWindowLong* to *SetWindowLongPtr*.

For more information on these topics, go to the 64-bit Windows Overview available at http://www.microsoft.com/windows2000/guide/platform/strategic/64bit.asp or see the Intel publication "Getting Your Code Ready for the IA-64 Architecture" available at http://developer.intel.com/ design/ia-64/getsoftready. For more information on APIs in particular, see the URLs at the end of this white paper.

Why Migration Tools May be Necessary

As in most upgrades, the effort in making the actual code changes is trivial when compared to the effort of identifying where the changes are to be made and their impact on other code. Take for example the job of resolving the assignment or comparison between pointer variables and int or long variables. First, you find and examine every existing int, long, and pointer variable to determine where a change is needed. Then, taking an iterative approach, you use your target compiler to identify and correct size conflicts. This process involves numerous cycles for a typical application because every change correcting one problem can inadvertently introduce new problems.

By its nature, the process also involves a high frequency of conflicts. For example, changing an int to 64 bits to support a comparison with a pointer could affect the parameters of a function and all other code that calls that function. Similarly, changes to a data structure or function parameter can affect other executables related by IPC, RPC, or data file interfaces. Changes can also affect applications built for other platforms that share structure definitions. Yet another problem involving circular changes can arise when the volume of code is large or when multiple developers are working on the code simultaneously.

To address such problems, you may want to consider using the MigraTEC Migration Workbench 32-64 WIN Upgrade for the Microsoft Windows* Operating System, which is highlighted in the following sections. (You can learn more about MigraTEC, Inc., the company that makes the Migration Workbench, at http://www.migratec.com.)

Using the MigraTEC Migration Workbench

The MigraTEC Migration Workbench is divided into three processes: Inventory, Analysis and Remediation. By their simplest definitions, Inventory parses the source code, Analysis identifies required code changes and Remediation makes the changes. Note that although these processes are most useful during the code-migration step of your upgrade process, they also can play a helpful role in the two earlier steps, assessing complexity and developing a plan.

Inventory Process

True to its name, the Inventory process assembles a hierarchy of source code and linked components through which you can identify the contents of your application, locate the source files and libraries to be incorporated, pinpoint any third-party dependencies and specify constraints required for compatibility with other applications. Inventory supports multiple build configurations using different files and libraries — even a mix of compiler versions — and it provides an automated way to log the contents of a build.

Designed to help locate missing source or header files, Inventory also provides reports documenting the content of your migration project. This makes Inventory an ideal tool for simplifying the process of assessing complexity and, in part, developing your migration plan.

Analysis Process

The output of Inventory is used by the Analysis process to identify and categorize invalidated cast operations, changes necessary to migrate from the ILP32 data model to the P64 data model, direct conflicts resulting from assignment of 64-bit pointer variables to 32-bit int or long variables, and other problem areas.

Analysis works by identifying each type or API that may need changing and provides you a choice of solutions. In some cases, Analysis selects the appropriate 64-bit change and requests your confirmation. In other cases, Analysis provides you the opportunity, through a source viewer and decision-support framework, to select your own change.

In every case, Analysis uses data flow propagation to help you to determine the impact of each data type change. This process follows the flow of data through pointer references, passed parameters, function returns, and structure members to help you to determine what new code, if any, might cause a conflict when you introduce a change. Where necessary, Analysis then gives you the opportunity to change a given variable to either a size variant type (for example, int_ptr or long_ptr) or an alternative of your choice. You can decide when you want Analysis to perform data flow propagation: either when each code change is made, or after you have decided on all code changes to be made.

Based on the information Analysis derives from data flow propagation, you can select appropriate solutions for as many levels of changes as you like, without having to

perform numerous recompiles. Because you are prescribing choices that are stored within a project file, rather than as edits to source code, you can easily backtrack for the purpose of selecting a different solution.

Remediation Process

After you run the Inventory and Analysis processes, the Remediation process uses the choices you stored in your project file to generate new source. Remediation can replace old headers, APIs, constants and types with the new target-interface constructs. In addition to direct replacements and parameter reordering, Remediation can provide new local variables for pass-by-pointer differences, new header includes and one-to-many call sequences with parameter distribution.

Remediation makes each change with exactly the same code constructs and preserves the original source-code layout and content. Remediation also audit-tags each change with a comment and inserts optional explanatory comments.

Once Remediation has implemented the changes, you can inspect, compile, link and test the edited code for all source files, all applications or components and all configurations. If you encounter problems, you can repeat the Inventory, Analysis and Remediation processes. For most applications, you will probably repeat the processes a number of times until your new source code compiles, links and tests to your satisfaction.

In Summary

While there are a number of benefits in upgrading existing C/C++ applications from Win32 to Win64, there is also some effort involved. For small, relatively simple applications that you will be upgrading on your own, you may find it reasonable to implement code changes by hand. However, for most other applications, especially those being upgraded by a team of individuals, a code migration tool is essential.

The MigraTEC Migration Workbench 32-64 WIN Upgrade for the Microsoft Windows Operating System is one such tool. With its Inventory, Analysis and Remediation processes, the Migration Workbench provides you a flexible and systematic approach to determining the necessary code changes and the impact of those changes on your application. By automating what is otherwise a tedious and error-prone process, the Migration Workbench will significantly reduce your migration time and enhance the quality of your upgraded applications.

For More Information

To learn more about preparing code for migration, see “Cleaning Code for the IA-64 Architecture” at http://developer.intel.com/update/archive/issue23/stories/top1.htm or the Intel “IA64 Self-Paced Tutorials” at http://developer.intel.com/vtune/cbts/ia64/index.htm. For more information on MigraTEC, Inc., go to http://www.migratec.com.